**Guided Study: MongoDB Aggregation Pipeline in Compass**

Notes about connection timeout:

I had serious issues with query stages not executing because of timeout. Sometimes I could delete the stage and enter it again, and it would run. I showed you in the lecture how to increase the max time for a stage, but that does not change the connection timeout limit. I found this advice[[1]](#footnote-1), and it seems to help:

* append this string after /test in your connection string when you connect to your cluster:

&keepAlive=true&poolSize=30&autoReconnect=true&socketTimeoutMS=360000&connectTimeoutMS=360000

* you can also create intermediate results as views or use the $out operator to create permanent collections (see last page of this handout). This allows you to see the size of the intermediate result (and you can use it to continue building the pipeline.)

Data Sets

You will utilize 3 real-world data sets for your mini-project. They were constructed for different purposes by different organizations, but can be combined to provide an example of real-world data augmentation and integration via querying across data sets.

**Collection 1:**

The *restaurants* data set is the first collection that you should already have it loaded in your MongoDB Atlas cluster from last week’s activities.

**Collection 2:**

I looked at <https://www.data.gov/> (there are over 210,000 data sets) to find NYC restaurant inspections. The data set I selected is described here:

<https://catalog.data.gov/dataset/dohmh-new-york-city-restaurant-inspection-results>

The data set is a large CSV files. It may take 5-10 minutes to upload. Here is the link for the data set:

<https://drive.google.com/open?id=1hcpCgkfMjVUktkueSRqp4bQ2Z2G6xHQR>

**Collection 3:**

I looked at NYC OpenData to find a third data set that I could link with restaurants and inspections:

<https://opendata.cityofnewyork.us/data/>

I looked at the city government data sets (there are more than 600 of them) and picked this one that gives demographics by zipcode for participants in a youth and community development program:

<https://data.cityofnewyork.us/City-Government/Demographic-Statistics-By-Zip-Code/kku6-nxdu>

Here is a link for the data set:

<https://drive.google.com/open?id=1ceEsbhQQ3OYEpgUoE8YsQUd9OBX7w_eW>

**Activities (do these in order)**:

1. Upload the data sets and store them in MongoDB as separate collections named

* restaurants
* inspections
* demographics

How many documents are in each of your collections?

Restaurants – 25,400

Inspections – 384, 400

Demographics – 237

2. How many top-level keys are there in

* inspections? 20
* demographics? 25

In the following steps, create one pipeline with successive stages in the given order.

1. Create a $match stage on the *restaurants* collection using the following filter:

"address.street": /Avenue$/,

"name": /^K/,

$expr: {$lte: [ {$toInt: "$address.zipcode"}, 10500 ] }

1. How many documents are in the resulting collection?

118

1. Create a $lookup stage joining with *inspections* (on *restaurant\_id* and *CAMIS*, respectively):

from: 'inspections',

localField: 'restaurant\_id',

foreignField: 'CAMIS',

as: 'restInsp'

1. Filter the collection to remove restaurants with no inspections:

restInsp: {$ne: []}

1. How many documents in the resulting collection?

61

1. Create groups on the cuisine and borough and count the number of instances of each group:

\_id: {cuisine: "$cuisine", borough: "$borough"},

cuisine: {$first: "$cuisine"},

borough: {$first: "$borough"},

count: {$sum: 1}

1. How many documents in the resulting collection?

365

1. Remove the embedded object from each document using projection:

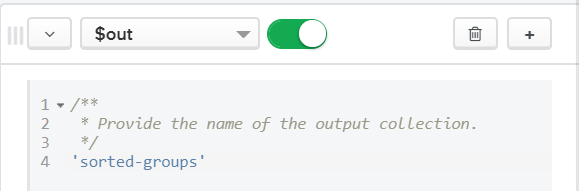
\_id: 0

1. Sort on *borough* and *cuisine* in ascending order.

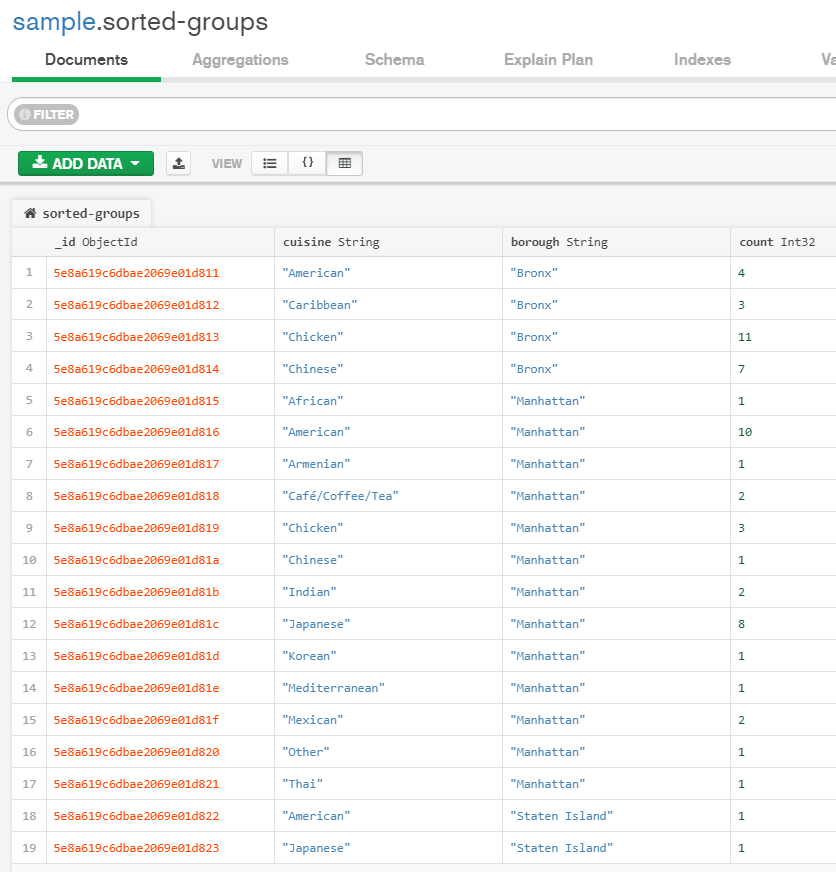
borough: 1,

cuisine: 1

Here’s a special bonus operator that is not in the lecture notes:



I used this to create a permanent collection (rather than a view) as the last pipeline stage. Here’s what the result looks like in the tabular view:



1. https://bobcares.com/blog/mongodb-connection-timeout/ [↑](#footnote-ref-1)